# Issue 036: Innovation

Graham Lee

September 6th, 2021

Welcome to the thirty-sixth issue of *De Programmatica Ipsum*, closing our third year of publication, and dedicated to the subject of *Innovation*. In this edition:

- Adrian discusses Microsoft's *de facto* monopoly of innovation around JavaScript[1].
- Graham studies the causes and effects of the fascination around the concept of innovation[2].
- In the Library section[3], Adrian dissects the good parts of Douglas Crockford's "JavaScript: The Good Parts"[4].

Enjoy this issue! Please subscribe to our free newsletter[5] to stay updated about new releases, share the articles on social media, or contribute[6] if you would like to support our work.

Cover photo by Diz Play[7] on Unsplash[8].

[1] https://deprogrammaticaipsum.com/innovationscript/
[2] https://deprogrammaticaipsum.com/innovation-to-what-end/
[3] https://deprogrammaticaipsum.com/category/library/
[4] https://deprogrammaticaipsum.com/douglas-crockford/
[5] https://deprogrammaticaipsum.com/newsletter/
[6] https://deprogrammaticaipsum.com/contribute/
[7] https://unsplash.com/@dizplay?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[8] https://unsplash.com/s/photos/innovation?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

# InnovationScript

Adrian Kosmaczewski

September 6th, 2021

If there is one galaxy in the software development universe that has suffered from the relentless, unstoppable, frantic, and unbearable pace of innovation, that one is, undoubtedly, JavaScript.

Around 2012, it was not uncommon to hear developers on Monday mornings talking over coffee about the new JavaScript framework they tried over the weekend. Some adventurous ones would even write yet another article in Medium or dev.to about their findings, with a nice comparison table that would help others find their way, or at least cover the hosting costs of their website via advertising. Some would even try to write their own.

## Frameworks, Frameworks, Frameworks...!

JavaScript took around 15 years to rise, from its inception to the boom of the 2010s. It all began to explode when Douglas Crockford[1] found out that JavaScript had some good parts; from that point on, the world of JavaScript frameworks evolved dramatically.

The first issue to tackle was the cross-browser chasm. Prototype[2], jQuery[3], MooTools[4], YUI[5], Ext.js[6], Dojo[7], they all provided ways to work around this horrible developer experience that was dealing with Internet Explorer 6 and later. Long before Apple introduced the

[1] https://deprogrammaticaipsum.com/douglas-crockford/
[2] http://prototypejs.org/
[3] https://jquery.com/
[4] https://mootools.net/
[5] https://github.com/yui/yui3
[6] https://en.wikipedia.org/wiki/Ext_JS
[7] https://dojo.io/

`<CANVAS>` object, Walter Zorn gave us the first DHTML drawing library[8]. The rise of Ruby on Rails pushed script.aculo.us[9] to the spotlight, providing the first glance at programmable animations. Actually, scratch that; you could do cross-browser, JavaScript-powered, timeline-based animations with Macromedia Dreamweaver back in 1998. But I digress.

Then came Ajax[10], a term famously coined by Jesse James Garrett[11] and whose initial implementation was based... on a Microsoft COM+ component called `XMLHttpRequest`. Gotta love those capitalization rules.

Later came Underscore[12] and Moment.js[13], to provide a better APIs for common operations, respectively for functional programming or date manipulation.

All of these libraries were the pinnacle of the "Web 2.0" craze, providing developers for the first time a tool with which to build a decent user experience across all browsers with, wait for it, a single codebase. You heard right. No more `if (userAgent == 'Netscape')` and stuff like that.

Talk about innovation.

Then server-side JavaScript became a thing– again. The ground-breaking Node.js[14] (itself based on a new wonder thing called V8[15], part of Google's efforts to eat the web) begat Express.js[16], and its middleware architecture inspired from Sinatra[17], itself inspired from a subset of Ruby on Rails; and now replicated by almost all REST API frameworks out there, in all programming languages and flavors.

On the command line front, Node.js brought us npm[18], and its hellish package management fury; apparently yarn[19] is better, but nobody can tell for sure. I wonder what is the package manager of hell called these days.

I defy any of the readers of this magazine to try to `npm update` your complicated Express.js application written in 2014 and never updated since, and try to fix all of the dependencies so that the tests run again in 2021. Innovating in the field of package managers made our apps weaker and impossible to update, but of course, indefinitely rewritable.

(Well, to be honest the same problem applies to apps written –and not often updated– in Swift 1.0, Android Jelly Bean, ANSI C 89, .NET 2.0, PHP 5.3... good luck with any of those. Maybe this is the future the Agile Manifesto authors were thinking about when they wrote it. Maybe not. Move fast and break things, kids.)

Then the iPhone and Android came along.

After Joe Hewitt's iUI[20] there were more sophisticated attempts to turn a website into an app. jQuery Mobile[21], Sencha Touch (of which this author has the doubtful honor of hav-

[8] http://walterzorn.de/en/jsgraphics/jsgraphics_e.htm
[9] https://script.aculo.us/
[10] https://en.wikipedia.org/wiki/Ajax_(programming)
[11] http://www.jjg.net/about/
[12] https://underscorejs.org/
[13] https://momentjs.com/
[14] https://en.wikipedia.org/wiki/Node.js
[15] https://en.wikipedia.org/wiki/V8_(JavaScript_engine)
[16] https://expressjs.com/
[17] http://sinatrarb.com/
[18] https://www.npmjs.com/
[19] https://yarnpkg.com/
[20] https://www.iui-js.org/
[21] https://jquerymobile.com/

ing written a book[22] about), and lately Ionic[23]. Around 2009 PhoneGap happened, later renamed as Apache Cordova[24], featuring a badly spelled anglicised version of the name of a Spanish city. Then Apple came out with the HTML5 Application Cache, but it turned out to be a douchebag[25]. For a while it seemed that the Financial Times[26] was the only team in the world able to build a decent web-based mobile app.

Finally, at some point Facebook vomited React Native[27], which quickly became the pinnacle of absolute madness and botched user experiences. Oh, but it is cross-platform, or so they say.

Lasciate ogni speranza[28], voi ch'entrate. In hindsight Responsive design[29] turned out to be a better option that any and all of those enumerated above.

The sacrosaint trinity of HTML, CSS and JavaScript ended up living all together in a same file thanks to Vue.js[30] and React[31], and apparently also Cycle.js[32] and whatnot. Innovation in 2015 meant loading 12 MB of JavaScript to be able to fill a login form.

Because filling HTML templates on the client is apparently more scalable than doing it on a server, hence we bothered all users with our single page applications that crippled our web experiences[33] for a whole decade.

Finally, on the desktop, we have got Electron[34]. Let us not say more.

## Microsoft

By the time this magazine saw the light, nobody was writing JavaScript anymore, of course; after the successful reception of CoffeeScript[35], Microsoft decided that it was time to own JavaScript once and for all, and created TypeScript[36], a surprisingly useful tool, and Visual Studio Code[37], a text editor written in and for TypeScript. Now they also own npm[38], GitHub[39], and it all feeds GitHub Copilot[40], no matter which license you were using.

And now, unbeknownst to most JavaScript developers, they are all using Microsoft technologies day in, day out. Quite an ironic turn of events, and one of the explanations for the skyrocketing MSFT ticker price in the last decade.

Among my readership, few, if any, will remember the HTML Application[41] and the HTML

[22] https://www.oreilly.com/library/view/sencha-touch-2/9781449339371/
[23] https://ionicframework.com/
[24] https://cordova.apache.org/
[25] https://alistapart.com/article/application-cache-is-a-douchebag/
[26] https://labs.ft.com/2011/06/ft-launches-first-major-html5-mobile-news-app/
[27] https://reactnative.dev/
[28] https://en.wikipedia.org/wiki/Inferno_(Dante)
[29] https://alistapart.com/article/responsive-web-design/
[30] https://vuejs.org/
[31] https://reactjs.org/
[32] https://cycle.js.org/
[33] https://how-i-experience-web-today.com/
[34] https://www.electronjs.org/
[35] https://coffeescript.org/
[36] https://www.typescriptlang.org/
[37] https://code.visualstudio.com/
[38] https://itsfoss.com/microsoft-npm-acquisition/
[39] https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/
[40] https://copilot.github.com/
[41] https://docs.microsoft.com/en-us/previous-versions/ms536495(v=vs.85)

Components[42] specifications created by... Microsoft for Internet Explorer 5. In 1998. Microsoft even came up with CSS filters[43] before GPUs were a thing.

Clearly, Microsoft has had its eyes laid on JavaScript for a long, long time.

In particular, HTML Components and Applications allowed for the creation of true desktop applications using the saint trinity of web development, all in a single file. You could, of course, substitute JavaScript with VBScript, a tasteless subset of Visual Basic which was even more crippled than its parent.

But the final result was surprisingly simple, and dare I say, extremely effective. Use HTML to add widgets to the screen of your application; style them with CSS; sprinkle some JavaScript to animate the inanimate. Thanks to COM+ components, developers could do pretty much anything with it. Want to read or write data to an Excel spreadsheet? There is a COM+ component for that. Want to send e-mail? There is another COM+ component for that, too.

Of course this was also a nice vector for worms, trojans, and viruses, starting with the infamous ILOVEYOU.txt.vbs[44] frenzy of 2000, but that is another story.

Microsoft was, by the way, also a pioneer in the field of JavaScript in the server; back in 1996 you could write whole Active Server Pages (ASP) with JavaScript (or rather, JScript[45]). "Oh but it was not asynchronous" I hear you say, and I say maybe (and frown at you), but you could serve tens of thousands of users simultaneously from a single Pentium II server with 32 MB of RAM. Talk about innovation in carbon emissions[46].

There has not been much innovation in the JavaScript arena since those times; at best, we had other players and some refinement. Paraphrasing Steve Jobs and his instructions per watt[47] analogy to justify his choice of Intel CPUs over PowerPC ones, web servers these days are arguably *serving less users per watt* than in 1996; at best, the same amount.

And most of them are mining cryptocurrencies anyway, not even actually serving users anymore, and the Earth, or what is left of it, is painfully feeling it.

## JavaScript, The Disappearing Parts

These days, the innovation around JavaScript is centered around making JavaScript go away. This trend was started, as mentioned above, by CoffeeScript and later by TypeScript; these days, you hardly need to write any JavaScript by hand at all. You can use Kotlin[48] and a hundred other languages to create a JavaScript application without writing JavaScript.

Little by little, JavaScript became the new assembly of the 2010s.

And quite literally so, this assembly analogy was taken to the extreme when Firefox announced WebAssembly[49], an actual assembly language based on a subset of JavaScript. Somebody then used it to run Doom on a browser[50]. Doom. Compiled from C++. Into WebAssembly.

[42] https://en.wikipedia.org/wiki/HTML_Components
[43] http://users.fred.net/dhark/demos/css/css_filter_examples.html
[44] https://en.wikipedia.org/wiki/ILOVEYOU
[45] https://en.wikipedia.org/wiki/JScript
[46] https://deprogrammaticaipsum.com/the-twenty-year-computer/
[47] https://en.wikipedia.org/wiki/Performance_per_watt
[48] https://kotlinlang.org/docs/js-overview.html
[49] https://webassembly.org/
[50] http://www.continuation-labs.com/projects/d3wasm/

Now we can create a full WebAssembly applications using Rust[51], compiling directly to WebAssembly from the confort of your command line, enjoying all the compile-time checks of the Rust compiler. You can even interact with the DOM if needed. And the weirdest of all is that you can even use Rust to re-create Flash[52], if so desired. The circle is closed. Neither Adobe nor Steve Jobs[53] saw that coming.

(Flash, which used a dialect of JavaScript called ActionScript[54], to entice developers into their own galaxy.)

Once written and compiled, you can run microservices written in WASM[55] inside your Kubernetes cluster; no need to create your web services with Express anymore. They are probably very fast and probably very difficult to debug; highly experimental, hence perfect for your project with a hard deadline that would be better served with single HTML page with a PHP page behind.

All is new again. Nihil sub sole novum. JavaScript is still there, somehow, but nobody can see it for sure. Hopefully somebody can still debug it. Or maybe not being able to debug it is part of the spec, who knows. At least you can get certified in JavaScript[56] these days. Whatever that means.

All of these innovations have rendered obsolete the "View Source" command in your browser. Ever tried to read the minified JavaScript in a page minified with Babel.js[57] or Gulp.js[58]? Forget about WebAssembly code unless you are a CPU designer.

Call me nostalgic, but I draft websites with Vanilla JS[59] or Postmodernize[60], the greatest web frameworks ever made, and arguably, the only one needed by 99.99% of web apps out there, allowing us to manage the DOM like we used to[61]. There are lots of tutorials[62] and guides[63], and even a full academy[64] to go back to lean, small, simple websites. Who knows, server side rendering[65] might be cool again soon.

It used to be a good practice to design websites that worked properly when JavaScript was disabled. These days, such an idea is not only laughable, it is actually counterproductive, and would render most of the web instantly and absolutely unusable. Even `XMLHttpRequest` is gone now; use `fetch()` instead[66].

We need more innovation in the field of innovation itself. So far, we have been reinventing (and arguably, worsening) things that worked properly before, while many actual issues remain untouched. In particular, JavaScript does not need any more abstraction layers built on top of it.

[51] https://www.rust-lang.org/what/wasm
[52] https://ruffle.rs/
[53] https://en.wikipedia.org/wiki/Thoughts_on_Flash
[54] https://en.wikipedia.org/wiki/ActionScript
[55] https://docs.krustlet.dev/howto/wasm/
[56] https://openjsf.org/
[57] https://babeljs.io/
[58] https://gulpjs.com/
[59] http://vanilla-js.com/
[60] https://postmodernize.telnet.asia/
[61] https://htmldom.dev/
[62] https://javascript.info/
[63] https://vanillajsguides.com/
[64] https://vanillajsacademy.com/
[65] https://css-tricks.com/servers-cool-once-again/
[66] https://fetch.spec.whatwg.org/

Ironically, now that Microsoft owns JavaScript *de facto*, this wish might as well come true. Embrace, extend, extinguish[67]. We will talk about this subject next month.

Cover photo by Mathew MacQuarrie[68] on Unsplash[69].

[67] https://en.wikipedia.org/wiki/Embrace,_extend,_and_extinguish

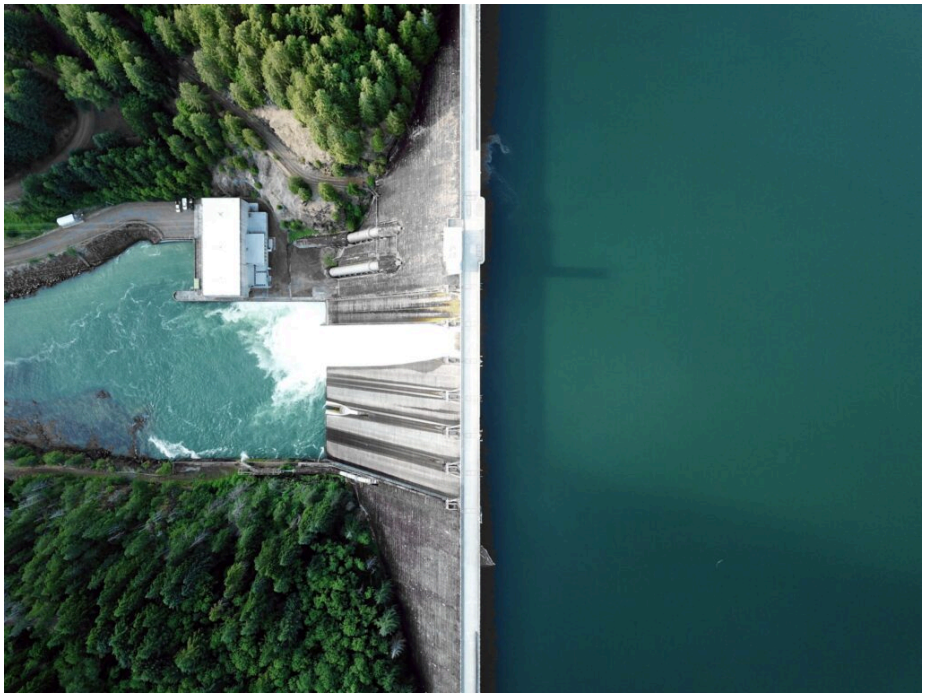
[68] https://unsplash.com/@matmacq?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[69] https://unsplash.com/s/photos/curse?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

# Innovation To What End?

Graham Lee

September 6th, 2021



This fascination society has with innovation is a funny thing. There is no need for it, and yet here we are. Always be disrupting. Move fast and break things.

You might be forgiven for thinking that it is some intrinsic part of capitalism: that we always need to have new things for the money machines to keep going around. In fact it is not. Growth capitalism—the idea that we always need more things—and consumer capitalism—the idea that you should always aspire to own more things—are definitely long-standing features of the modern hypercapitalism that developed in the second half of the twentieth century and that holds the United States, United Kingdom, European Union and other states in its thrall today.

In earlier times, money-makers were willing to settle for other sources of growth: mercantilism and colonialism. No need to go to all that trouble to invent new commodities if you can just discover and exploit new sources elsewhere. No need to make work more efficient if you can pick up new workers by the boatload in one country and force them to work for subsistence (or less) in another.

No, innovation was originally a dirty socialist trick. How liberating would it be if the work of sixty hours could be done in forty, or twenty, or five? If the work of a hundred teams of ten workers each could be replaced by the work of five cyberneticians and eight robotics engineers?

By getting better at work, the socialists could reduce the number of people needed to do the work, and the time for which they had to do it. By taking the lessons in manufacturing and

machinery learned by the British capitalists in the steam revolution, they could turn steam to the benefit of the people. Thus we have Trotsky castigating the Stalinist regime in The Revolution Betrayed[1] for only raising industrial production by 8.5% in 1932, "instead of the 36% indicated by the year's plan". Congratulating the worker for bringing the USSR to the number one producer of tractors and sugar in the world, and decrying Stalin for failing to see the value of the (cancelled) Dnieperstroy hydroelectric station.

So focused were the Soviets on innovation and industrial productivity that the idea became an object of parody. In his allegorical novel Animal Farm[2], George Orwell describes Boxer the horse, happy and proud to work for the regime but ultimately to overdo it and work himself to death. Boxer was based on Soviet workers called Stakhanovists, workers who regularly put in overtime or otherwise exerted themselves to surpass their quotas and "fulfil the five year plan in four years!"

Orwell was, by the way, anti-totalitarian but no anti-socialist. In fact he saw socialism as good politics, but often saw socialists as its least capable promotors. "To sum up," he concludes The Road to Wigan Pier[3], a reflection on working-class life in the North West of England, "There is no chance of righting the conditions I described in the earlier chapters of this book, or of saving England from Fascism, unless we can bring an effective Socialist party into existence."

But this would require intelligent propaganda. "Less about mechanical progress, tractors, the Dnieper dam and the latest salmon-canning factory in Moscow"; all those things Trotsky and Stalin had been arguing over.

OK, so innovation is a socialist issue. What has that got to do with computers? Well, the history of the computer is the history of the Cold War. Soon after the conflict known here as the Second World War and in the USSR as the Great Patriotic War was over, and the three buddies Churchill, Stalin, and Eisenhower had divided up the remains of Europe and the Pacific theatre, the Allied Powers decided that only profit-loving allies could be in the club.

Gone from the official histories was the importance of the German rout at the Battle of Stalingrad (for which streets all over Europe are still named today); in was Britain Stands Alone. Last week's BFF was this week's public enemy #1.

Luckily, all those computers we had been making would come in handy in the new conflict. The USSR had been using the same Lorenz encryption system as the Nazis: no need to tell them that Bletchley Park had developed the Colossus machines which could crack their code. The ENIAC was no longer working out how to lob nukes at Berlin, but what about Moscow?

It became important to beat the reds, and to beat them at any game going. The Olympic games and the space race are two obvious proxy battlegrounds. Economics was a fairly easy win: when one side has a currency and the other a government-approved scrip, any currency-based metrics are going to be easily gamed to favour the marketeers.

What is going to be trickier, however, is beating the socialists at their own game: productivity. How do you out-do the salmon-canning, Dnieper-damming, "from each according to their ability" ideologues at the game of getting more out of each than their abilities? How do you out-industrialise a nation that went from a peasant-farming backwater in 1905 to the winner of the technological war in 1945?

[1] https://www.marxists.org/archive/trotsky/1936/revbet/revbetray.pdf
[2] https://libcom.org/files/animal_farm.pdf
[3] https://libcom.org/files/wiganpier.pdf

Thus was innovation – already, without a doubt, present in the pre-war but post-industrial revolution western civilisation – pushed into the centre of everybody's economic and industrial policies, and turned all the way up to 11.

And what has it got us? You can now order any product made by stroking a pocket glass and have it next day or even quicker. What are you doing with all that time that is freed up?

Certainly not leisure. The average hours worked per week in developed countries has certainly dropped since the industrial revolution but remained roughly constant since 1980[4]. Tracking all that work in Jira has not led to doing less of it.

But if you are working more, it is not helping. Productivity has stagnated[5] since computers were introduced. We are not canning more salmon per labour-hour than we did before we could save meeting minutes to Confluence.

All we are getting for all this innovation is the "opportunity" to innovate some more.

Cover photo by Dan Meyers[6] on Unsplash[7].

[4] https://ourworldindata.org/working-hours

[5] https://deprogrammaticaipsum.com%20https://austinvernon.eth.link/blog/softwareisprocess.html

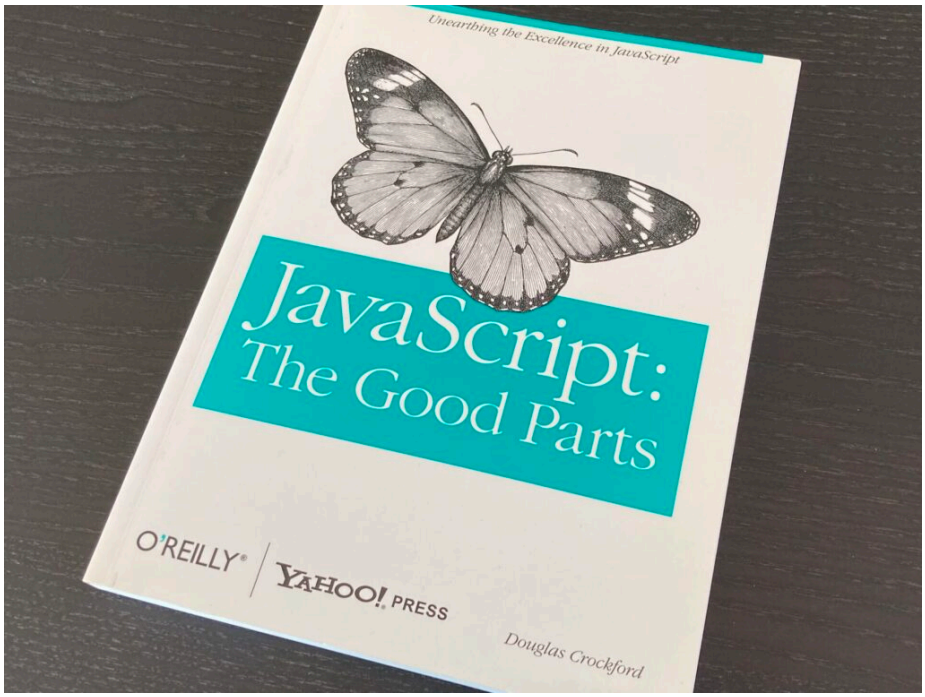
[6] https://unsplash.com/@dmey503?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[7] https://unsplash.com/?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

# Douglas Crockford

Adrian Kosmaczewski

September 6th, 2021




O'Reilly published in 2020 the seventh edition of one of the biggest bestseller programming books of the past 25 years, Flanagan's "JavaScript: The Definitive Guide"[1]. At 700 pages and weighing 1.2 kg, it is a book that easily stands out in any good programmer's library. Many developers have used such information to joke about the fact that the good parts of JavaScript, as catalogued by Crockford in his eponymous 2008 book[2], is merely 180 pages long, and weights only 290 grams; that is, only 25% of JavaScript is actually any good.

That comparison is misleading, to say the least, particularly because Crockford's book also includes a colophon, a dedication, a preface, lots of syntax diagrams, and two appendices called "Bad Parts" and "Awful Parts". All in all, the good parts are just 140 pages, give or take.

Before the book, Crockford was already a celebrity for many of us dealing with JavaScript in a daily basis. This author came across his humble web page[3] around 2002, still available at the time of this writing, where he claimed that JavaScript was "The World's Most Misunderstood Programming Language". The revelation was not small, and in hindsight, was a major event for the software industry.

The issues with JavaScript, of course, start with the name. The language that Brendan Eich baptised LiveScript was renamed to JavaScript during the dotcom craze of the 1990s, because marketing and NASDAQ and Java and IPOs and Sun and Netscape and whatnot. These

[1] https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016/
[2] https://www.oreilly.com/library/view/javascript-the-good/9780596517748/
[3] https://www.crockford.com/javascript/javascript.html

days LiveScript is an actual different thing[4], but still related to JavaScript, just to make things more confusing for everyone. But the industry remained confused about the subject of Java vs. JavaScript for decades. Jeremy Keith summarized the situation[5] in one phrase: "Java is to JavaScript as ham is to hamster."

Then there were paradigm issues; Java was the apotheosis of object orientation. JavaScript... has a `new` keyword, but one which does not work exactly the way a Java developer would expect. And it has a weird[6] inheritance model. And of course object orientation in the 1990s was all about inheritance and nothing about composition, because few had read Barbara Liskov's paper[7] yet. So if JavaScript's inheritance was not in the flavor of Java's or C++'s, clearly it was inferior. To illustrate the confusion, Crockford even explained how to encapsulate private fields[8] with JavaScript in his website.

Finally, there were quirks. There were many[9]. JavaScript could use semicolons, or not; there were `String` and `string`, and nobody quite understood the relationship between both; there was `==` and `===` and nobody quite understood the difference between both. And the list could go on and on and on.

(These were the subjects developers argued about in Slashdot[10] or Evolt[11] back in those days, long before Reddit or A List Apart were a thing.)

Crockford's website[12] (and later, book) was arguably one of the igniters of the renaissance of JavaScript. In the wake of the dotcom bubble crash[13], as web developers were struggling to find a job, Crockford realized that what we needed was to calm down for a minute, have a cup of tea, and realize that JavaScript was *different*. Just that; not better, not worse, just different, and that no, it did not suck[14]. We just needed to take a breath, and stop screaming the words "startup" and "IPO" to every venture capitalist walking in the street.

If this was not enough for his résumé, let us not forget that Crockford also introduced JSON[15] and JSLint[16] to the world.

But we can go even further, without exaggeration or hyperbole: in retrospective, the biggest contribution of Crockford was to finally make functional programming mainstream. It was this book, together with the rise of Ruby on Rails, what would make millions of developers all over the world, including the author of these lines, understand what a closure was for the first time. And since, according to Herb Sutter at least, the free lunch was over[17], the time was ripe for functional programming to rise.

Cover photo by the author.

---

[4] https://livescript.net/
[5] https://www.smashingmagazine.com/2009/07/misunderstanding-markup-xhtml-2-comic-strip/
[6] https://www.crockford.com/javascript/prototypal.html
[7] https://deprogrammaticaipsum.com/barbara-liskov/
[8] http://www.crockford.com/javascript/private.html
[9] https://quirksmode.org/
[10] https://slashdot.org/
[11] https://evolt.org/
[12] https://www.crockford.com/javascript/
[13] https://en.wikipedia.org/wiki/Dot-com_bubble
[14] https://www.sitepoint.com/interview-doug-crockford/
[15] https://www.json.org/json-en.html
[16] https://en.wikipedia.org/wiki/JSLint
[17] http://www.gotw.ca/publications/concurrency-ddj.htm